Argonne NATIONAL LABORATORY

Learning Python

Getting results for beamlines and scientific programming

Advanced Python: Using arbitrary parameter lists

The Advanced Photon Source is an Office of Science User Facility operated for the U.S. Department of Energy Office of Science by Argonne National Laboratory

U.S. DEPARTMENT OF ENERGY | Office of Science

Outline of topics to be covered

1. Review
   - simple (positional) argument passing
   - Using keyword arguments in place of positional arguments
   - Defining optional parameters
2. Receiving adjustable numbers of parameters
3. Receiving unspecified keyword parameters
4. Combined parameter receiving
5. Passing a list/tuple to define positional parameters
6. Passing a dict to define keyword parameters
7. Combined parameter sending

Review of Parameter Passing

- The parameters for a function is most commonly a sequence of variables:

```
def Funct(p1, p2, p3):
    return p1+10*p2+100*p3
```

- The function is then called by passing values or variables for those parameters

```
>>> Funct(1,2,3)
321
```

```
>>> a = b = c = 2
>>> Funct(a,b,c)
222
```

Review: Parameter Passing by Keyword

- One can also pass parameters by keyword, or even in combination, but the correct minimum number is required

```
>>> Funct(p3=1,p1=2,p2=3)
132
>>> Funct(1,p3=2,p2=3)
231
```

Review: optional parameters

- One can specify a default value for the final parameters for a function:

```
def Funct(p1, p2=2, p3=0):
    return p1+10*p2+100*p3
```

- Nothing changes if values are specified for all parameters

```
>>> Funct(1,2,3)
321
```

- But the defaults are used if a value is not specified

```
>>> Funct(1)
21
>>> Funct(1,p3=2)
221
```

- Parameters without defaults must come first in the argument list and must be specified in calls:

```
>>> Funct()
TypeError: Funct() takes at least 1 argument (0 given)
```

More complex usage: arbitrary parameter lists and accommodating arbitrary keyword parameters

- You may have noticed that some Python functions take different numbers of parameters (for example zip):

```
zip( ['a','b','c'], [1,2,3], [4,5,6], )
```

  - Or

```
zip( [1,2,3], [4,5,6], )
```

- This can be done in your own code, if convenient

- You can even define a routine that will accept arbitrary keyword parameters.
  - Why do this? If you write a wrapper around a routine in an external package, there may be new functionality added – you don't want to have to keep updating your code just to pass new arguments.
  - You will see this used often in wxPython

Receiving an adjustable numbers of parameters

To accommodate an arbitrary number of parameters place a parameter with a ***single*** * in front it into the argument list:

```
def Funct(*parms):
    print parms
    sum = 0
    for i in range(len(parms)):
        sum += parms[i] * 10**i
    return sum
```

- When this is done, all non-keyword parameters are placed in a tuple referenced by the argument variable:

```
>>> Funct(1)
(1,)
1
>>> Funct(1,2)
(1, 2)
21
>>> Funct(1,2,3,4,5)
(1, 2, 3, 4, 5)
54321
```

Receiving unspecified keyword parameters

- To define a routine that will accept arbitrary keyword parameters place a parameter with a ***double*** asterisk (**) in front

```
def Funct(**dict):
    print dict
```

- This causes the keyword parameters to be loaded into a dictionary

```
>>> Funct(p3=1,p1=2,p2=3)
{'p2': 3, 'p3': 1, 'p1': 2}

>>> Funct(1,2,3,4,5)
TypeError: Funct() takes exactly 0 arguments (5 given)
```

Using mixed parameter passing

- All of these parameter modes can be combined in a single function definition:

```
def Funct(p1, p2=0, *args, **kw):
    print 'p1=',p1
    print 'p2=',p2
    print 'args=',args
    print 'kw=',kw
```

```
>>> Funct(1)
p1= 1
p2= 0
args= ()
kw= {}
```

```
>>> Funct(1,2,3,4,test='?')
p1= 1
p2= 2
args= (3, 4)
kw= {'test': '?'}
```

- Note that one parameter (p1) is required, but the rest are optional
- p1 will be the 1st positional parameter, p2 the 2nd.
  - args/kw are empty if extra positional/keyword parameters are not used

Passing positional parameters from a list/tuple

- It is also possible to supply a list or tuple as set of positional parameters using a similar syntax:

```
L = (1,2,3,4,5)
Funct(*L)
```

Is equivalent to

```
Funct(1,2,3,4,5)
```

- As we saw before, when combining positional (non-keyword) and keyword parameters when calling a function, the positional parameters must come first
  - Positional parameters must come before keywords

```
Funct(*L, p1=0)
```

  - If mixing lists and individual parameters the list must come last:

```
Funct(0, 1, *L)
```

Passing keyword parameters parameters

It is possible to pass keyword parameters from a dictionary, by passing the dictionary name, prefixed with two asterisks (**)

```
>>> D = {'p1':1, 'p3':3, 'p2':2, }
>>> Funct(**D)
p1= 1
p2= 2
args= ()
kw= {'p3': 3}
```

Combining positional and keyword passing

One can pass parameters by all four methods combined, but that the positional parameters must come first and the dict (**D) must come last

```
>>> L = (1,2,3,4,5)
>>> D = {'p4':4, 'p3':3, 'p5':5, }
>>> Funct(6,p9=0,*L, **D)
p1= 6
p2= 1
args= (2, 3, 4, 5)
kw= {'p3': 3, 'p9': 0, 'p4': 4, 'p5': 5}
```

- keyword and list-supplied parameters can be supplied in either order:

```
Funct(6,*L, p9=9, **D)
```

```
Funct(6, p9=9, *L, **D)
```

- Beware of double-defining parameters when using a dictionary and positional parameters:

```
>>> Funct(6,**D)
TypeError: Funct() got multiple values for keyword argument 'p1'
```

Argument pass-through

- If one writes a function with adjustable positional and keyword arguments, the variables can be used to pass all arguments through

```
def MyWrapper(*L, **KW):
    do_stuff()
    Wrappee(*L, **KW)
```

All arguments supplied to `MyWrapper` end up getting passed to `Wrappee`

- This is most commonly used in classes where arguments supplied to an overridden method are supplied to the overridden function (for example `__init__`)

Summary

- Python has four methods for passing parameters:
  - Positional
  - Keyword
  - Positional from List/tuple
  - Keyword from dict
- Python has four ways for a function to accept parameters.
  - Keyword with fixed position (value required)
  - Keyword with fixed position and default value
  - Optional positional
  - Optional keyword

The ability to pass positional arguments by tuple/list and the ability to receive positional arguments by tuple, as well as the keyword implementation with dicts need not be used together on both the function definition and call, but commonly are.